

A Scientific Approach to Process Improvement in Agile Processes

MSE Studio Reflection Paper

Michael Keeling

10/21/2009

1. Introduction

Nearly all software development processes provide for improvement through regular retrospection. Agile processes are no different. Many agile processes prescribe techniques intended to facilitate team discussions and prompt team members to raise process concerns during retrospection meetings [1]. For these techniques to be successful, team members must be experienced, observant, interested, and process savvy. When any one of these components is missing from a team, agile retrospection loses its effectiveness. While many might argue that simply following an agile process such as Extreme Programming “by the book” should be sufficient, agile processes are in use on a variety of projects decreasing the likelihood that assumptions made within the process will hold in so many different situations.

At the same time, a more systematic, data-driven approach to understanding how a process is working for a team might undermine the many positive benefits of using an agile process. This paper proposes that by applying the scientific method to process improvement, agile teams can remain lightweight while simultaneously approaching process improvement in a systematic, predictable, and repeatable way. Carrying out small experiments helps constrain data collection for process improvement and allows teams to overcome shortfalls in experience, awareness, interest, and common sense. Experimentation in turn helps teams better understand how specific parts of a process are working for the team and results expose clear recommendations on how to tailor processes.

This paper will outline the how the Square Root team used the scientific method to evaluate the merit of pair programming on an XP project. Section 2 provides a brief summary of the SQUARE Project and describes the author’s and team’s experience with agile software development in this context. Section 3 discusses the current state of process improvement techniques in agile processes while section 4 outlines the need for a more systematic approach. Section 5 describes the Square Root team’s experience applying the scientific method as a means for process improvement. Section 7 enumerates lessons learned from the project while section 8 concludes.

2. SQUARE Project Overview

The SQUARE project was completed as part of Carnegie Mellon University’s Master of Software Engineering (MSE) program [2]. The project consisted of five team members, all students in the MSE program, and a single client. The client was a senior member of the technical staff at the Software Engineering Institute (SEI). The MSE studio uses a capstone element, the Studio Project, which allows students to practice concepts that were learned through coursework in a realistic project setting. The Studio Project lasts for the duration of the program; projects are assigned during orientation and conclude shortly before the end of student’s final semester.

While the project was conducted in an academic setting, the client was considered a paying customer and the students were all experienced engineers. Team members began the program with between two and five years experience with varied technical backgrounds including quality assurance, data analysis, project management, and development on desktop applications, web applications, and real-time distributed systems. The project lasted four semesters (16 months) with varying time commitments from 9 hours to 48 hours per week depending on the semester. The project had a strict timeline, a budget of 4,800 engineering hours, and was monitored by two senior faculty members of the MSE program.

2.1 Methodologies

During the construction phase of the project, the team used Extreme Programming (XP) [3]. No team members had prior experience with XP however most had used at least some of XP's practices in their previous jobs. At least one team member had experience with each of the following: Test Driven Development, refactoring, pair programming, creation and use of a coding standard, continuous integration, and incremental development. Team members also had experience with a variety of other software processes, most notably the Personal Software Process (PSP), the Team Software Process (TSP), Scrum, and the Rational Unified Process. Process evaluation and improvement techniques were introduced through core courses and electives taken by some members of the team. These courses tended to be metrics-focused and data-intense in their treatment of process evaluation.

2.2 Product

SQUARE (Security Quality Requirements Engineering) is a nine step process for eliciting security requirements [4]. The purpose of the SQUARE project was to create a web-based tool that would shepherd SQUARE adopters through all nine steps in the process, help teams using SQUARE select and execute methods complementary to SQUARE, and provide requirements analysis capabilities. The developed tool will be used by the SEI to assist researchers working on SQUARE, for educational purposes (e.g. interactive demonstrations, presentations, workshops), and distributed to business customers who wish to use SQUARE on real projects. The project's client is the principal investigator for SQUARE.

3. Process Improvement in Agile Processes

Regular reflection and incremental process improvement are core tenants of agile processes including XP [5] [3]. XP relies on iteration retrospectives in which the team collectively brainstorms about which practices worked well and which practices did not. Other retrospective tactics often employed include simple devices meant to prompt team members' memories about what happened during the project and how they felt about iteration activities [1]. Based on these reflection methods it is apparent that XP retrospectives are generally qualitative in nature and tend to rely on individual team members' informal observations and intuition. Further, since XP only prescribes data collection in a few key areas it is generally impossible to qualitatively assess how an XP team is performing overall [6].

Team members' experience and personal interests play a significant role in determining what is identified during a retrospective and the specific actions taken to correct detrimental behaviors, gaps in understanding among the team, and problems with the development process. When deciding what to do, teams generally rely on democratic decision making strategies. While this usually ensures that the team buys in to the decisions made, democratic decision making can result in suboptimal decisions; decisions where what the team wants to do and what needs to be done may not be aligned. Generally, teams tend to focus on the current sources of pain, the processes most obviously in need of immediate improvement, often at the sacrifice of longer term gains.

Even when teams agree on solutions, it is often difficult for agile teams to identify specific courses of action for achieving desired improvements. Depending on a team's commitment to process, explicit tasks for improvement may never be created, relying instead on promises made by team members to change behavior. Agile teams in particular suffer from an inability to reliably articulate the root cause of problems due to the lack of data for analysis. Teams and organizations practicing XP can at best achieve

CMM level 3 (processes are defined and repeatable) [6] and at worst may never consistently repeat results due to gaps in understanding over how the process is actually working for the team.

4. The Need for Controlled Process Improvement in Agile Processes

Agile processes are in use on a wider array of projects than ever before: small and large [7], co-located teams and distributed teams [8], simple systems and complex systems. Iterative practices such as those advocated in XP are becoming standard practice for many organizations and have even been included in United States Department of Defense standards [9]. With such a wide range of organizations adopting agile processes for such a diverse range of projects, teams can no longer afford nescience concerning the relationship between the team and the process it uses. While XP may work “out of the box” for many teams, the recent rise in popularity of agile processes increases the likelihood that teams using agile will not conform to the ideal conditions required for XP practices to work effectively. Process tailoring is often required.

Guidance for tailoring agile processes in industrial settings remains scarce as research struggles to keep up with practice [10]. While some inroads have been made in assessing the efficacy of XP using case studies [11], teams are largely left to their own devices for determining whether, what, and how to tailor XP. At the same time, since these projects are business-focused, industrial teams don’t have many of the luxuries afforded academia for measuring practices in controlled experiments and analyzing the results before choosing the best course of action.

At the same time, industrial XP teams must somehow overcome shortcomings in experience by avoiding an overreliance on intuition for guiding process improvement. Relying exclusively on intuition, as is done on most agile teams, creates risks for wasting project resources through inappropriate tailoring. Practices that should be improved may go unnoticed and other practices that are working may be abandoned due to misguided perceptions.

A scientific approach to process improvement similar to that used by the Square Root team can help overcome these issues by constraining process metrics and using measures to support intuition in an agile fashion.

5. Team Square Root Approach to Process Improvement

The Square Root team adopted 10 of the 12 practices advocated by the first edition of XP [3]. Square Root’s client was not on site and the team had spent the previous semester developing a system architecture using the Architecture Centric Design Methodology [12]. The team was initially unsure about the use of pair programming given the fixed deadlines and limited resources for the project.

At the end of the second iteration, it was impossible for the team to determine whether pair programming was efficient enough for this team to complete the project on time or what level of software quality pairs were producing with any confidence. To make matters worse, the team was passionately divided about pair programming, some loving it, others hating it. To better understand how pair programming was working for the team, in terms of both efficiency and quality, the team decided to conduct a series of experiments. The experiments were intended to help the team make an objective decision about whether to continue pair programming or tailor XP.

The Square Root team used the scientific method to help decide what to do about pair programming. The process used by the team is shown in Table 1.

Table 1 Square Root applied the scientific method to help answer questions about pair programming.

Step in the Scientific Method	Square Root's application to pair programming
Ask a question	Is pair programming efficient enough to finish the project on time with the desired level of quality?
Do background research	Research by Williams et. al. [13] and team projections indicates that the schedule will be close with pair programming. Research by Phongpaibul and Boehm [14] indicates that pair programming and Fagan inspection are about the same in terms of quality. These results may not be applicable to Square Root since it more closely resembles an industrial team than an academic team.
Construct a hypothesis	The Square Root team should be able to approximately reproduce the results discussed in [13] and [14].
Test the hypothesis by doing an experiment	Use the Goal Question Metric approach [15] to identify metrics for assessing the hypotheses. Identify data necessary to calculate those metrics and create means of collecting the data if necessary. Identify and mitigate project risks introduced by the experiment. Divide work into experiment categories so that each category is approximately equal, creating an informal replicate study [16].
Analyze your data and draw a conclusion	Analyze collected data and calculate metrics. Use conclusions drawn from metrics and data to make recommendations for process improvement or back intuited observations about process. Present learned lessons from the experiment.
Report your results (was the hypothesis correct?)	Discuss the results with the team in terms of the hypotheses as a part of the regular iteration retrospective. The retrospective should include a mixture of data-backed and intuition-based recommendations. As a team, determine what process changes, if any, need to be made.

6. Experience with Scientific Data Collection and XP

Based on the results from previous academic studies [14] [13], the team developed a series of hypotheses.

Hypothesis 1: Pair programming produces code of similar or better quality than individual programming with Fagan inspection.

Hypothesis 2: Pair programming requires 15 – 80% more effort to complete a feature than individual programming on features of similar size.

Hypothesis 3: Pair programming requires 60 – 80% less calendar time to complete a feature than individual programming on features of similar size.

Treating these hypotheses as goals, the team applied the Goal Question Metric approach [15] to identify metrics necessary for measuring the outcome of the experiment. Data collection methods were then devised so the team could calculate the metrics. The following metrics were identified for each of the hypothesis under test.

Hypothesis 1 Metrics

- Number and type of defects per KLOC discovered during Fagan inspection.
- Number and type of defects per KLOC discovered during pair programming.
- Number and type of defects discovered during system testing.

Hypotheses 2 and 3 Metrics

- Total hours spent during development of a feature by activity. Activities, as defined by the team, were assigned to each task and included such things as design, bug fixing, coding (new development), and refactoring.
- Total hours spent working in pairs for features developed using pair programming.

Since XP does not give any specific guidance for classifying defects, the team decided to borrow defect classifications from the Team Software Process [17]. To measure defects discovered during pair programming a simple tally sheet was created in which the current co-pilot would mark issues raised during development (see Figure 1). The tally sheet was traded for the keyboard and mouse during a pair programming session. Tally sheets were collected at the end of each day and collated into a database.

Programmer 1: Marco Estimate (hours):
 Programmer 2: Michael Estimate (hours):
 date: 7-16-2009 Start Time: 5:36 End Time: 7:39
 Milestone/Feature: Create Project Task Points: (Low/Medium/High)
 Task Description: Dialog Boxes

Data / Documentation	Syntax	Build / Package	Assignment	Checking	Interface	Function	System	Environment
Comments, coding standard/style	Things that don't compile	CM, file maintenance, package stuff	Declarations, duplicate names, scope	Business rule violations, verify inputs	I/O, Method calls and references	Logic, algorithms	Performance, Timing, Java, design	Compile, test, hot keys, support
	///		///	///				

Instructions: Record defects in real-time according to the type of defect detected by marking the appropriate column. The current co-pilot should record the defects. This paper should be traded for the keyboard and mouse between the pair.

Figure 1 Tally sheets used to record issues discovered during pair programming sessions. Each tick mark indicates an issue discovered by the current copilot.

Effort data was recorded for each task in the team's SharePoint server. Two identical entries would be made for tasks undertaken in pairs, one entry for the task owner and the other entry for the partner. The task owner was responsible for ensuring that data was correctly recorded in the SharePoint server. Copilot tasks were marked as such in the task database. Other than the more rigorous requirements for handling paired tasks, this procedure is identical to data collection methods used prior to the experiment.

The team used use case points [18] to estimate the size of features on the product backlog. For planning purposes, use case points were also used to estimate the approximate effort required to build each feature. Features were then assigned to an experiment category, *paired*, *individual*, or *mixed*, based on the number of points. Features in the *mixed* category were not considered for the experiment. Experiment categories each had approximately the same number of features of different sizes and approximately the same number of total points.

Since XP is a release-driven process with short iterations (two weeks in Square Root's case), the team was concerned about potential negative consequences due to conducting the pair programming experiment. To mitigate experiment risks, the team conducted the experiment over three iterations, taking features from all three categories each iteration. Since the team allowed the client to prioritize features at the beginning of iterations, substitutions were made between experiment categories for

features with identical point values to ensure iterations had a reasonable mix of features from each experiment category. The team also identified a risk trigger for abandoning the experiment. If planned features from one category could not be released on schedule for two consecutive iterations, the experiment was to be terminated to get the project back on schedule.

Table 2 This table shows the planned allocation of use cases to be implemented for the experiment. The Integration Proof of Concept was eventually removed from the experiment since the chosen design was too different to be reasonably comparable to other development work. Square Lite was also eventually dropped as it was not prioritized during the planning game before the end of the project.

Individual Use Cases	Points	Paired Use Cases	Points
Choose Project	5	Agree on Definitions	5
Choose Step	5	Close Step	5
Create/Edit Inspection Technique	5	Collect Artifacts	5
Delete Project	5	Create Project	5
Reset Password	5	Square Lite	5
Register New User	9	Export Data	9
View/Edit Use Profile	10	Prioritize Requirements	9
Inspect Requirements	14	Integration Proof of Concept	14

At the conclusion of iterations, experiment data was made generally available to the team for analysis. The analyzed data was used during the team's iteration retrospective meeting. Team iteration retrospective meetings used a mix of data-backed and intuition-based recommendations for process changes. With regards to pair programming, recommendations based on the identified hypotheses were made in addition team discussion about pair programming.

Before the team began the experiment, the experimentation process was recorded in the team wiki for easy access. In addition to the experiment procedure, hypotheses, GQM triples, experiment category definitions, defect categories, and the initial feature allocation were also recorded. As questions concerning experiment procedures were resolved, the answers were recorded and shared among the team. Identified risks were recorded in the team's risk database as done in previous iterations.

6.1 Experiment Results and Observations

Preliminary experiment data was used during iteration retrospectives and a more thorough reflection was performed following the conclusion of the experiment. The team identified several interesting trends through the experiment, a summary of which is shown here. Because the experiment consisted of only a single sample (the Square Root team), these results should not be taken generally and are only applicable to the Square Root team.

Hypothesis 1 was proved. While the results of this test were not statistically significant as determined by an independent T-test, code produced using pair programming resulted in slightly fewer defects per KLOC than code written individually and inspected. It is also interesting to note that code produced using pair programming had more predictable quality as a function of size. This is likely because programming in pairs overcomes individual biases in experience, mood, concentration, or other factors that might cause quality to differ when programming alone.

Hypothesis 2 was disproved. Pair programming required a range between 11% and 40% less effort than individual programming with Fagan inspection. Additionally, pair programming required a range of 26%

less and 12% more effort than individual programming without Fagan inspection. The team was quite surprised by this discovery. Though the team is confident in the results, again, the results were not found to be statistically significant by an independent T-test. The general implication of these results is more important; the team tends to be highly efficient when pairing thus requiring only a small increase in overall effort when working alone without inspection. Of course, without inspection no quality benefits can be realized.

Hypothesis 3 was proved. For the Square Root team, pair programming required 54 – 62% less calendar time to complete a feature than individual programming with Fagan inspection. Additionally, as hypothesized, pair programming required 60 – 80% less calendar time to complete a feature than individual programming alone. Since the hypothesis was only interested in individual programming (without Fagan inspection), this hypothesis stands true. Again, an independent T-test revealed these results to not be statistically significant.

Based on these observations, the team concluded that pair programming was vital to completing the project within the time constraints. The team also noticed several other interesting observations about pair programming as implemented by the Square Root team. Since pair programming produced code of more predictable quality, the number of injected defects can be more readily predicted simply by knowing the program size. The team also observed that pair programming never occurred more than 95% of the time for any given feature from the paired experiment category. Paired milestones in the experiment averaged 85% paired time while milestones outside of the experiment averaged only 60% paired time. This is important since pair programming is often touted as an alternative to Fagan inspection capable of delivering 100% code coverage. This idea has been shown to be more urban legend. Finally, the main focus when paired was on writing working code while the focus during Fagan inspection was on finding defects. As a result of this, another general observation was that different knowledge was shared among the team during pair programming than during inspections.

At the conclusion of the experiment, the team had greater confidence in pair programming and decided to continue the practice “unenforced” as before the experiment. Some results will be used for planning in the future. The general consensus, as shown by the data, was that pair programming created code that was “good enough” in terms of quality, but pair programming is not a true substitute for inspection from a knowledge sharing perspective. The real-time issue tracking sheets were both fun and interesting and the team chose to continue using them as a way of keeping the current co-pilot engaged and to monitor the effectiveness of pairs over time.

7. Lessons Learned

Following the conclusion of the construction phase of the studio project, the Square Root team reflected about the project during a facilitated postmortem. The postmortem revealed several insights about the use of the scientific method as a means for quantifying process improvement and the experiment itself. Some of these insights are presented as recommendations in this section.

7.1 Experiment Champion

Two experiment champions emerged early in the project, one presenting discovered research on pair programming and encouraging the team to try an experiment, the other enforcing the experiment procedures. These individuals were responsible for planning and overseeing the experiment as well as educating the rest of the team on the hypotheses and procedures including how to record data. Since

one of the experiment champions was the XP Coach, it was easy for the champions to closely work together to determine how best to carry out the experiment without damaging other processes.

7.2 Experiment Plan

As the experiment progressed, having an easily accessible, recorded experiment plan became essential. The plan was most important during the first few days of the experiment and when analyzing the collected results. The written experiment procedure guided the team as members became accustomed to data collection changes. Additionally, since detailed decisions concerning interpretation of the results had been recorded in the plan, the team was not tempted to bias analysis once the data was collected. One example of information recorded before the experiment was how much paired effort was required for a feature to be considered written using pair programming.

7.3 Data Collection

The Goal Question Metric approach helped the team to clearly identify required metrics for determining whether the hypotheses were achieved or not achieved. Once the metrics were known, identifying the minimum set of data necessary to calculate the metrics was relatively easy. The team was able to avoid problems by making small modifications to existing data collection methods and using a combination of physical (e.g. real-time issue tracking sheets) and electronic (e.g. SharePoint, Bugzilla) data collection methods. Incorporating adjunct data collection methodologies such as the Team Software Process can help overcome data shortcomings in XP.

7.4 Team Agility

XP asks that teams value working code over documentation and embrace change. Generally, this creates a volatile environment, hostile to scientific experimentation. The Square Root team overcame this problem by implementing the system using both the control method (individual programming) and experimentation method (pair programming) simultaneously. This was only possible because the team used an estimation proxy, use case points, for iteration planning. Abstracting specifics about the experiment plan allowed the team to embrace XP principles in the experiment and while still drawing meaningful conclusions.

7.5 Experiment Risk Management

The team used explicit risk management practices [19] to identify conditions created by the experiment that could have a negative impact on the outcome of the project. The last thing the team wanted was for the experiment to delay the release of software. Mitigation strategies for identified risks included the use of physical paper for collecting real-time pair programming issue data, implementing features using both the control and experimental methods simultaneously, recording the experiment plan in the team wiki, and clearly budgeting the experiment's duration. While the mitigation strategies were effective in preserving agility, achieving experiment objectives, and enabling project completion, experiment results were delayed. Specifically, simultaneously implementing features paired and individually increased the number of iterations necessary to gather sufficient data to reach conclusions on all parts of the experiment. Further, focusing on a single hypothesis might have allowed the team to reach conclusions sooner. Explicit understanding of the risks introduced by the experiment was a key to the success of both the experiment and the project.

7.6 Hawthorne Effect

The Hawthorne Effect is a phenomenon in which subjects participating in an experiment alter their measured behaviors as a result of knowing they are being studied [20]. In practice, subjects experience temporary productivity boosts when they know they are being observed. The Square Root team, as the

subject for these experiments, was likely no different. This is especially true in the case of more invasive observational techniques such as the real-time pair programming issue sheets used by the Square Root team. To the greatest extent possible, existing or automatic observation techniques should be used to counteract artificial productivity boosts and preserve the long term applicability of the experiment. While it should not be relied upon, experiments might be used to increase team attention in specific areas of a software project. Since the team chose to continue using some of the observational techniques such as the real-time issue sheets, the findings from the experiment will still be applicable.

7.7 Observations beyond Hypotheses

While scientific hypotheses formed the starting point for the experiment, driving procedures and data collection methods, the team relied on lessons learned in addition to scientific results. These observations were possible thanks to the nature of experimentation. Two practices, the control and variable (in this case individual programming with Fagan inspection and pair programming), were examined side-by-side making it easy to perform a pair-wise comparison. As an example, one team member noted after a Fagan inspection, “I feel like I am more focused on finding issues during inspections than I am while pair programming.” Another team member noted that “I hate going back and fixing the issues we discover during inspections. The issues come up days after I’ve banged out the code and it takes me forever to remember the context.” These kinds of insights are just as important as the scientific results and can help explain results. Qualitative observations should not be ignored even with scientific experimentation. These observations were brought up during iteration retrospectives.

7.8 Statistical Significance

The Square Root team’s experiments combined replication and lessons learned validation models [21] to reduce costs and maintain team agility. Since the results were only intended to be applied to the team, application beyond the team is inconsequential. Combining replication with the lessons learned allowed the team to make conclusions based on trends and observations with only a few data points. Recognizing that these experiments were controlled only through informal replication allowed the team to execute the experiment inexpensively and quickly, yielding results in a timely fashion so they could be used within the short XP iteration cycles. Formal experiment replication would not have allowed this and would have been overkill in this scenario. In fact, none of the data collected was found to be statistically significant for comparison byway of individual T-tests. Given the extremely small sample size, this is not surprising. In spite of this, the data was good enough to demonstrate promising general trends and helped the team understand how it was operating in specific areas. Since the ultimate goal was to determine whether the team should continue using pair programming or not, statistical significance was not important.

8. Conclusions

Agile processes without tailoring rely too heavily on team members’ intuition, experience, and passing observations. In addition, with agile being applied across so many domains, in so many new ways it is not enough to simply believe that agile practices will work for all teams in all situations. Due to the gap between research and industry practice, it is up to teams to determine what, how, and when to tailor agile processes. This cannot be done effectively by relying exclusively on qualitative observations.

The increased overhead of data collection necessary to quantitatively measure process improvements increases costs and alters the inherent behavior of agile processes. However, as shown in this paper, scientific experimentation acts as a constraining force against unfettered data collection, in essence

creating opportunities for incorporating more structured, qualitative measurements into agile processes without reducing the agility of the process. In this way, inexperienced teams or teams with unobservant team members or members lopsidedly focused on only certain process areas, reduce process improvement risks similarly to other, more heavy-weight, metrics-driven processes such as the Team Software Process.

The experience presented in this paper strongly advocates for a simultaneously controlled and informal, scientific approach to process improvement but not at the sacrifice of qualitative retrospective practices. Both informal replication and lessons learned validation models should be used. Scientific experiments are a useful tool but probably is not necessary when the team is experienced or when project conditions align with known research. Of course, if the team is curious or unsure, an experiment can extend great confidence to the team and even temporarily boost productivity through the Hawthorne Effect. If in doubt, put it to the test.

Science is a method of discovery. Teams should consider running experiments if for no other reason than to learn more about the processes they use and how they work in addition to the numerous benefits listed here. This sort of intrinsic curiosity is healthy for teams and might even strengthen work in other engineering areas. And given the gap between research and industry, publishing experiment results in the form of whitepapers and blog posts would be positive for the software industry as a whole.

9. Acknowledgments

I would like to thank my fellow Square Root teammates (Sneider Sequeira, Marco Len, Yi-Ru Liao, and Abin Shahab) for their help on the SQUARE project and for their participation in these experiments. I would also like to thank our studio mentors, Dave Root and John Robert for their guidance in setting up the experiments.

10. Works Cited

- [1] E. Derby, D. Larson, and K. Schwaber, *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf, 2006.
- [2] D. Garlan, D. P. Gluch, and J. E. Tomayko, "Agents of Change: Educating Software Engineering Leaders," *IEEE Computer*, vol. 30, no. 11, pp. 59-65, Nov. 1997.
- [3] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
- [4] N. R. Mead, E. D. Hough, and T. R. Stehney II, "Security Quality Requirements Engineering (SQUARE) Methodology," Software Engineering Institute, Pittsburgh, Technical Report CMU/SEI-2005-TR-009, 2005.
- [5] K. Beck, et al. (2001) Agile Manifesto. [Online]. <http://www.agilemanifesto.org/>
- [6] M. C. Paulk, "Extreme Programming from a CMM Perspective," *IEEE Software*, vol. 21, no. 6, pp. 19-26, Nov. 2001.
- [7] A. Elssamadisy, "XP on a Large Project - A Developer's View," in *XP/Agile Universe*, Raleigh, NC, 2001.
- [8] M. Kircher, P. Jain, A. Corsaro, and D. Levine, "Distributed eXtreme Programming," in *Second International Conference on Agile Processes and eXtreme Programming in Software Engineering*, 2001.
- [9] C. Larman and V. R. Basili, "Iterative and Incremental Development: A Brief History," *IEEE Computer*, vol. 36, no. 6, pp. 47-56, Jun. 2003.
- [10] M. Lindvall, et al., "Empirical Findings in Agile Methods," in *XP/Agile Universe*, Chicago, IL, 2002, pp. 197-207.
- [11] L. Williams, W. Krebs, L. Layman, A. L. Antón, and P. Abrahamsson, "Toward a Framework for Evaluating Extreme Programming," in *Empirical Assessment in Software Engineering (EASE)*, Edinburgh, Scotland, UK, 2004, pp. 11-20.
- [12] A. J. Lattanze, *Architecting Software Intensive Systems: A Practitioner's Guide*. Boca Raton, FL, United States of America: Auerbach Publications, 2009.
- [13] L. Williams, R. R. Kessler, W. Cunningham, and R. Jefferies, "Strngthening the Case for Pair Programming," *IEEE Software*, vol. 26, no. 4, pp. 19-25, Jul. 2000.

- [14] M. Phongpaibul and B. Boehm, "A Replicate Empirical Comparison between Pair Development and Software Development with Inspection," in *First International Symposium on Empirical Software Engineering and Measurement*, Madrid, Spain, 2007, pp. 265-274.
- [15] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric Paradigm," in *Encyclopedia of Software Engineering*, J. J. Marciniak, Ed. Wiley, 1994, pp. 528-532.
- [16] B. A. Kitchenham, "Evaluating Software Engineering Methods and Tools," *SIGSOFT Software Engineering Notes*, vol. 21, no. 1, pp. 11-15, Jan. 1996.
- [17] W. S. Humphrey, *Introduction to the Team Software Process*. Addison-Wesley, 1999.
- [18] M. Cohn, "Estimating with Use Case Points," *Methods & Tools*, vol. 13, no. 3, pp. 3-13, Oct. 2005.
- [19] C. R. Nelson, G. Taran, and L. d. L. Hinjosa, "Explicit Risk Management in Agile Processes," in *Agile Processes in Software Engineering and Extreme Programming*, Limerick, Ireland, 2008, pp. 190-201.
- [20] R. McCarney, et al., "The Hawthorne Effect: a randomised, controlled trial," *BMC Medical Research Methodology*, vol. 7, no. 1, Jul. 2007.
- [21] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," *Computer*, vol. 31, no. 5, pp. 23-31, May 1998.